Framework for task scheduling in heterogeneous distributed computing using genetic algorithms

Andrew J. Page and Thomas J. Naughton

Department of Computer Science,
National University of Ireland,
Maynooth, Ireland.
andrew.j.page@may.ie,
http://www.cs.may.ie/∼apage

Abstract. An algorithm has been developed to dynamically schedule heterogeneous tasks on to heterogeneous processors in a distributed system. The scheduling strategy operates in a dynamically changing computing resource environment and adapts to variable communication costs and variable availability of processing resources. The scheduler utilises a genetic algorithm to minimise the overall execution time. Experiments are performed which show that the algorithm can achieve near optimal efficiency, with up to 100,000 tasks being scheduled.

1 Introduction

Scheduling heterogeneous tasks on to heterogeneous resources, otherwise known as the task allocation problem, is an NP-hard problem for the general case [7]. In multi-processor systems, such as a distributed system, one would expect linear speed-up when additional processors are employed to process tasks. Properties of a distributed system such as communication overheads, heterogeneous processors, and heterogeneous tasks can reduce the efficiency achieved by the system. The development of a scheduling strategy is required to produce schedules which seek to minimise the total execution time. The scheduler must also have the ability to adapt to varying resource environments.

Many heuristic algorithms exist for specific instances of the task scheduling problem, but are inefficient for a more general case [6, 15]. The use of Holland's genetic algorithms [13] (GAs) in scheduling, which apply evolutionary strategies to allow for the fast exploration of the search space of possible schedules, allows for solutions which seek to minimise the execution time to be found quickly and for the scheduler to be applied to more general problems. Many researchers have investigated the use of GAs to schedule tasks in homogeneous [4, 14, 23] and heterogeneous [1, 9, 10, 17, 22] multi-processor systems with great success.

Unfortunately assumptions are often made which reduce the generality of these solutions, such as that scheduling is calculated off-line in advance and cannot change, all communications times must be known in advance [1, 4, 9, 14, 22], networks provide instantaneous message passing [10, 23] and that processors

will always run at the same speeds [1, 4, 9, 10, 14, 15, 19, 21–24]. These assumptions limit the generality of these scheduling strategies in real world systems. We make no assumptions about the homogeneity of the processors, or about the availability of system resources.

In this paper a scheduling strategy is presented which uses a GA to schedule heterogeneous tasks on to heterogeneous processors to minimise the total execution time. It operates dynamically, allowing for tasks to arrive for processing continuously, it considers variable network contention and variable speed processors based on historical observation of system conditions, and it maximises the use of the processor which is responsible for scheduling tasks.

In Sect. 2 we review related work. In Sect. 3 we give an overview of how a GA operates. In Sect. 4 we describe our scheduling algorithm. In Sect. 5 we present the results of our performance experiments, and conclude in Sect. 6.

2 Related Work

There are many examples in the literature of artificial intelligence techniques being applied to task scheduling [1, 4, 9, 10, 14, 15, 17, 19, 21–24]. Meta-heuristic search techniques such as GAs [13], tabu [8], and ant colony search [3] are most applicable to the task scheduling problem because we wish to quickly search for a near optimal schedule out of all possible schedules. Good results have been yielded through the use of GAs in task scheduling algorithms [1, 4, 9, 10, 14, 15, 17, 19, 21, 23, 24].

Much work has been done on using GAs for static scheduling [1, 4, 9, 14, 22], where schedules are created before runtime, but the state of all tasks and system resources must be known a priori and cannot change. This limits these schedulers to specific problems and systems.

Dynamic GA schedulers [10, 17, 23, 24] create schedules at runtime, with knowledge about the properties of the system and tasks possibly not known in advance, allowing for variable system and task properties to be considered. Dynamic GA schedulers are thus the most practical of the two to use for real world distributed systems. Current dynamic GA schedulers have been shown to produce near optimal schedules in simulations, although assumptions that have been made limit their usefulness. Communications costs and the possibility of variable processing resources are not considered. We propose that historical information about communication costs and variable processor speeds be considered when creating a schedule. An algorithm is presented in this paper which corresponds to the real world and addresses properties which have not been previously addressed in GA based dynamic task scheduling algorithms.

3 Genetic Algorithms

A GA is a meta-heuristic search technique which allows for large solution spaces to be heuristically searched in polynomial time, by applying evolutionary techniques from nature [13]. GAs use historical information to exploit the best so-

lutions from previous searches, known as generations, along with random mutations to explore new regions of the solution space. A GA can be broken down into three steps important steps, selection, crossover, and random mutations. Selection according to fitness is a source of exploitation, and crossover and random mutations promote exploration.

A generation of a GA contains a population of strings, σ_i, each of which correspond to a possible solution from the search space. Each string in the population has a value associated with it, F_i, indicating how 'fit' the string is, or how good the string is, compared to the rest of the strings in the population.

4 Scheduling Algorithm

In this section we detail our scheduling algorithm which utilises the GA metaheuristic search technique. The algorithm we have developed is based on one developed by Zomaya *et al.* [23, 24]. We have created an algorithm which can adapt to varying resource environments and can produce near optimal schedules. The GA algorithm is only performed if there are more unscheduled tasks than processors; if there are fewer tasks than processors, the largest task gets assigned to the processor which will finish processing it earliest.

We wish to schedule an unknown number of tasks for processing on a distributed system with a minimum execution time. The processors of the distributed system are heterogeneous, and it is assumed we have non-exclusive usage of their processing resources. The available processing resources on each processor can vary over time. Processors can be added, removed, or fail, and can be idle. Each processor can be uniquely identified by a scheduling processor, which is dedicated to creating schedules to map tasks to processors. The available network resources between processors in the distributed system can vary over time. Each task, t_i, to be scheduled for processing has an associated processing resource requirement and the task can be uniquely identified. Tasks are also indivisible, independent of all other tasks, and can be processed by any processor in the distributed system.

Tasks arrive at unknown intervals for processing, and are placed in a queue of unscheduled tasks. Batches of tasks from this queue are scheduled on processors during each invocation of the scheduler. Each task has a resource requirement which is measured in millions of floating point operations per second (Mflop/s). Each processor can only process a single task at any one time. The available processing resources of each processor are known (in Mflop/s), measured using Dongarra's Linpack benchmark [5]. This is a recognised standard used to benchmark systems for inclusion in the list of Top 500 supercomputers [16]. Available processing and network resources vary over time, so the exponential smoothing function (see Sect. 4.1) is used to minimise localised fluctuations, thus allowing for a more realistic processing environment to be controlled. A single processor is dedicated to scheduling (as in [11]) although it is recognised that the scheduling algorithm itself could be distributed over all of the processors in the distributed system.

Each idle processor in the system requests a task to process from the scheduler, which is then processed and returned. The scheduler contains a queue of tasks which have been mapped to each processor, and when a request for work is received from a processor the task at the head of the corresponding queue is sent for processing. A processor does not contain a queue of tasks, because network resources are limited and processing resources are not dedicated, thus we do not wish to repeatedly hand out the same tasks multiple times when system resources change significantly.

4.1 Exponential Smoothing Function

An exponential smoothing function, $A_i = A_{i-1} + \nu \times (a_i - A_{i-1})$, is utilised to allow for a single 'average' value, A, to accurately represent multiple values, $a_1, a_2, .., a_n$, which arrive sequentially producing $A_1, A_2, .., A_n$, while smoothing out fluctuations, and allowing recent values to exert more influence than older values whose influence tends towards zero. The total number of values is N. The spread of the function is controlled by ν, where $\nu = [0, 1]$.

4.2 Communication costs

Communication costs, such as bandwidth (bits/second) and latency (seconds), between the scheduler and the processors available for work should be considered in any schedule produced. We use historical information about network communication times between processors to estimate future communication costs, thus allowing for a more real world scheduling environment to be considered.

The cost of sending a task t_k to processor P_j is $C_{k,j} = (b_{k,j}/Q_{k,j}) + (R_{k,j}/2)$. $C_{j,k}$ denotes the communication cost between processor j and the scheduler for task k. This value is derived by calculating the cost of sending and receiving messages. The latency of the communications channel from the scheduler to each processor is tested, and a round trip time is calculated which is included in R_j using the smoothed average function. When a task is sent from the scheduler to a processor, its size in bytes, $b_{k,j}$, is calculated, where k is the kth task sent. The processor then sends back an acknowledgement, and the task round trip time is noted as $rt_{k,j}$. The number of bytes sent per second is calculated as $q_{k,j} = (b_{k,j}/rt_{k,j)-R_j}$ and $q_{k,j}$ is then sent to the smoothing function to produce $Q_{k,j}$, which denotes the bandwidth of a given channel.

4.3 Dynamic Batch Size

Tasks arrive for processing at random intervals and are added to the queue of unscheduled tasks T. The queue may contain a large number of tasks waiting to be scheduled; however, it may take a long time to find a schedule for all the tasks which efficiently utilises system resources. Instead a dynamically sized batch considers batches of tasks for scheduling from the queue, which reduces the probability of processors waiting on the scheduler to finish creating a schedule.

A single processor is dedicated to scheduling (such as in [11]), so we strive to maximise the use of its resources, a cost which has not been considered by some dynamic scheduling algorithms [10, 17, 23]. It is easier to obtain a high utilisation of processing resources if there is a high ratio of tasks to processors [23].

The time a GA takes to run from start to finish is related to the size of the batch, kH^2, where k is a constant overhead and H is the size of the batch. Thus using the exponential smoothing function a smoothed average time, ST where $ST \geq 1$, for a single element in the batch can be calculated. We wish to fully utilise the resources of the dedicated scheduling processor. The time when the first processor becomes idle is calculated as follows, $minTime = \min_{j=1}^{M}(\delta_j / P_j)$, where δ_j is the processing time in Mflop/s waiting to be processed by processor P_j and M is the number of processors in the distributed system. Thus $H = \lfloor \sqrt{ST} \rfloor$, where $H \geq 1$, resulting in a dynamically sized batch which fully utilises the processor belonging to the scheduler.

Once a schedule has been assigned the batch size is once again recalculated and another schedule is produced until there are no more tasks in T to schedule.

4.4 Encoding

Each string, σ_i, in the population represents a different possible schedule. We have chosen to use double precision floating point numbers for each character of σ_i. Each character in σ_i provides information about mappings between tasks and processors. The number of characters in σ_i is $\omega = H + M - 1$, where H is the number of tasks in the batch, and $M-1$ is the number of partitions in the string, where M is the number of processors. Each task in the batch is mapped to a processor, with a unique task ID number identifying the task, and a delimiter separating the different processors queues.

4.5 Most into Least

An initial population is generated using the Most-Into-Least (MIL) list scheduling heuristic, which has been successfully used in other GA task schedulers [4, 10]. A random number of tasks, are assigned to processors in a round robin fashion. The remaining tasks are then sorted, using Quicksort [12], and allocated in a round robin fashion to the processors which will finish processing them the earliest, taking into account existing and assigned tasks for each processor. This leads to a well balanced randomised initial population.

4.6 Fitness Function

A fitness function attaches a numerical value to every string in the population, which indicates how much better one schedule is over the rest of the schedules in the population. We use relative error to generate a fitness value for each string (used in [10]) in the population because it allows for the makespan and load balancing of a schedule to be represented in a single numerical value. This is

only an internal metric to heuristically direct the exploration of the search space. When the GA has finished running, the string with the smallest makespan is used by the scheduler to assign tasks to processors.

In a given string σ_i, an error e_j is calculated for each processor, P_j, where e_j is a non-negative floating point number. Previously assigned, but unprocessed, load for each processor is considered by calculating δ_j, the finishing time of a processor j. The finishing time is calculated as $\delta_j = (L_j/P_j)$, where L_j denotes the previously assigned load, measured in Mflop/s, and P_j is the current processing power in Mflop/s of processor j.

The theoretical optimal processing time can now be found, $\psi = (\sum_{i=1}^{N} t_i / \sum_{j=1}^{M} P_j) + \sum_{j=1}^{M} \delta_j$ where t_i is the processing requirement of task i in the batch (in Mflop/s) and N is the total number of tasks in the batch.

The relative error of a string is given as $E_i = \sqrt{\sum_{j=1}^{M} |\psi - (L_{k,i} + \sum_{u=1}^{M} ((t_y/P_j) + C_{t_y,j}))|^2}$ where $C_{t_y,j}$ is the communication cost (see Sect. 4.2), of scheduling a task, t_y, on a processor j. The fitness value of a string is $F_i = 1/E_i$, where $F_i = [0, 1]$. A larger value indicates a better or fitter schedule.

4.7 Selection, Crossover and Mutation

We choose to use the standard weighted roulette wheel method of selection which is widely used by previous researchers who have applied GAs to task scheduling [4, 9, 10, 14, 19, 23]. Each string in the population, σ_i, is assigned a slot, ς_i, between 0 and 1. The size of a slot is $\varsigma_i = F_i \times (\sum_{j=1}^{\rho} F_j^{-1})$, where $\sum_{i=1}^{\rho} \varsigma_i = 1$. After the selection process is complete we use the cycle crossover method [18] to promote exploration as used in [23]. We have chosen to use two types of mutation to promote exploration of the search space. First of all we randomly swap elements of a randomly chosen string in the population. The next type of mutation aims to make a string more balanced. A random string is selected and a task on the processor, P_j, with the greatest relative error, $E_{i,j}$, is then randomly selected and inserted randomly into the queue of a different processor. This heuristic encourages well balanced solutions to be found in less time.

4.8 Stopping Conditions

The GA will keep evolving the population until one or more stopping conditions are met. The string with the lowest makespan is selected after each generation and if it is less than a specified minimum, the GA stops evolving. The maximum number of generations is set at 1000 [23]. The GA will also stop evolving if one of the processors becomes idle, in which case it will return the best schedule found so far.

5 Experiments

The scheduling algorithm described in Sect. 4 has been implemented and simulations have been performed, with up to 50 heterogeneous processors, and up to 100,000 randomly generated heterogeneous tasks. Each experiment was repeated a number of times and an average result was calculated for each point. We also implemented the original algorithm that our algorithm is based on, developed by Zomaya *et al.* [23], which is the current state of the art dynamic GA task scheduler for homogeneous distributed computing. It was easily adapted to work with heterogeneous processors by using Mflop/s as the measure of the rate of execution rather than time.

Tasks are scheduled across 50 heterogeneous processors with a processing resource range of 10 to 100 Mflop/s. We assume that all of the tasks arrive for processing at the beginning of the simulation, for these experiments. Determining a representative set of heterogeneous computing task benchmarks remains a challenge for the scientific community in this research area as noted by Theys *et al.* [20]. We have decided to generate random sets of tasks for scheduling using the Poisson distribution. We use randomly generated task sets because: we wish to demonstrate the algorithms effectiveness over a broad range of conditions, a set of heterogeneous computing benchmark tasks do not exist, and it is not clear what characteristics a 'typical' task would exhibit [20].

We have decided to use a population size of 10, which is known as a micro GA [2] and used in [10, 23, 24], which speeds up computation time without impacting greatly on the final result. We have also compared our scheduling algorithm against a number of well known batch and immediate mode heuristic schedulers. An immediate mode scheduler only considers a single task for scheduling on a FCFS basis.

The Min-min batch scheduler begins with a batch of unscheduled tasks. It then schedules the task with the minimum completion time to the next available processor. This is repeated until all tasks have been scheduled. The Max-min batch scheduler is similar to the Min-min scheduler except it schedules the task with the maximum completion time to the next available processor. The earliest first immediate mode scheduler considers tasks on a FCFS basis. When a task arrives for scheduling, it is assigned to the processor which will finish processing it the earliest. The lightest loaded scheduler is also an immediate mode scheduler. When a task arrives for scheduling it is assigned to the processor which has the lightest existing load.

5.1 Communication

We wish to show that our algorithm provides greater efficiency in a system with variable communication costs. To demonstrate its effectiveness we vary the ratio of the task processing requirement to communications costs, and measure the efficiency achieved. We fix the available processing resources and the size of the batch, to allow for the effect of communication costs to be demonstrated. We wish to schedule 100,000 tasks with a view to maximising the efficiency of

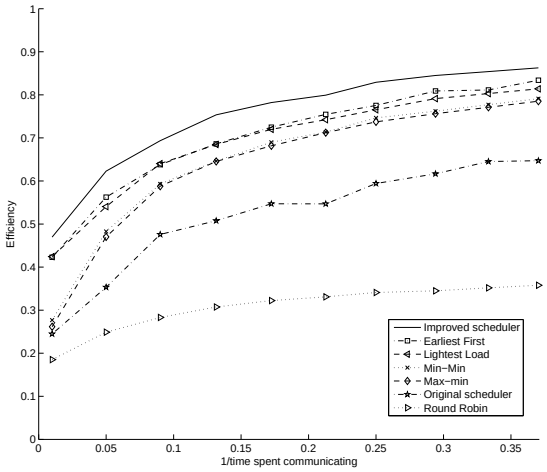

Fig. 1. Efficiency of schedulers varying communication to task size ratio

the processing resources in the distributed system. The communications costs between each processor and the scheduler are normally distributed.

Figure 1 shows that the improved algorithm proposed in this paper consistently provides schedules with greater efficiency over all of the other scheduling algorithms. The consideration of communication costs allows the improved scheduler to estimate a communications cost when creating a schedule, resulting in an overall improvement in efficiency of the scheduler.

6 Conclusion

A scheduling algorithm has been developed to schedule heterogeneous tasks on to heterogeneous processors in a distributed system. It provides efficient schedules, adapting to varying resource environments with respect to processing resources, and communications costs. The algorithm also fully utilises the dedicated processor running the scheduler. The GA employed the MIL list scheduling heuristic to create a well balanced randomised initial population. The fitness function utilises the relative error metric internally which promotes a well balanced solution with a low makespan. Roulette wheel selection is used to exploit past results to direct the search for efficient schedules. Cycle crossover promotes exploration of the search space. Random swaps and random re-balancing of processor queues within strings perturb the search and facilitate a better exploration of the search space.

Results have been presented which show that the algorithm proposed in this paper consistently uses processors more efficiently than the current state

of the art GA algorithms for the same problem. It is more suitable for real-world use because it considers properties of distributed systems, such as variable communication costs and variable speed heterogeneous processors, which other algorithms for the task scheduling problem do not consider.

7 Acknowledgement

Support is acknowledged from the Irish Research Council for Science, Engineering, and Technology, funded by the National Development Plan.

References

1. I. Ahmad, Y.-K. Kwok, I. Ahmad, and M. Dhodhi. Scheduling parallel programs using genetic algorithms. In A. Y. Zomaya, F. Ercal, and S. Olariu, editors, *Solutions to parallel and distributed computing problems*, chapter 9, pages 231–254. John Wiley and Sons, 2001.
2. A. Chipperfield and P. Flemming. Parallel genetic algorithms. In A. Y. Zomaya, editor, *Parallel and Distributed Computing Handbook*, pages 1118–1143. McGraw-Hill, New York, USA, first edition, 1996.
3. A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of the First European Conference on Artificial Life*, 1991.
4. R. Correa, A. Ferreira, and P. Rebreyend. Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 10(8):825–837, August 1999.
5. J. Dongarra, J. Bunch, C. Moler, and G. Stewart. *LINPACK Users Guide*. SIAM, Philadelphia, USA, 1979.
6. H. El-Rewini, T. G. Lewis, and H. H. Ali. *Task scheduling in parallel and distributed systems*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1994.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, 1979.
8. F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.
9. M. Grajcar. Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system. In *Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 280–285, New Orleans, Louisiana, USA, 1999. ACM Press.
10. W. A. Greene. Dynamic load-balancing via a genetic algorithm. In *13th IEEE International Conference on Tools with Artificial Intelligence*, pages 121–129, Dallas, Texas, USA, November 2001.
11. B. Hamidzadeh, L. Y. Kit, and D. Lilja. Dynamic task scheduling using online optimization. *IEEE Transactions on Parallel and Distributed Systems*, 11(11):1151–1163, November 2000.
12. C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.
13. J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
14. E. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, Feb 1994.

15. H. Kasahara and S.Narita. Practical multiprocessing scheduling algorithms for efficient parallel processing. *IEEE Transactions on Computers*, 33(11):1023–1029, November 1984.
16. List of the Top 500 Super Computers. http://www.top500.org.
17. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, November 1999.
18. I. M. Oliver, D. J. Smith, and J. Holland. A study of permutation crossover operators on the traveling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 224–230. Lawrence Erlbaum Associates, Inc., 1987.
19. H. J. Siegel, L. Wang, V. Roychowdhury, and M. Tan. Computing with heterogeneous parallel machines: advantages and challenges. In *Proceedings on Second International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 368–374, Beijing, China, June 1996.
20. M. D. Theys, T. D. Braun, H. J. Siegal, A. A. Maciejewski, and Y.-K. Kwok. *Mapping Tasks onto Distributed Heterogeneous Computing Systems Using a Genetic Algorithm Approach*, chapter 6, pages 135–178. John Wiley and Sons, 2001.
21. A. Y. Zomaya, M. Clements, and S. Olariu. A framework for reinforcement-based scheduling in parallel processor systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):249–260, March 1998.
22. A. Y. Zomaya, R. C. Lee, and S. Olariu. An introduction to genetic-based scheduling in parallel processor systems. In A. Y. Zomaya, F. Ercal, and S. Olariu, editors, *Solutions to Parallel and Distributed Computing Problems*, chapter 5, pages 111–133. John Wiley and Sons, 2001.
23. A. Y. Zomaya and Y.-H. Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(9):899–911, September 2001.
24. A. Y. Zomaya, C. Ward, and B. Macey. Genetic scheduling for parallel processor systems: comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed Systems*, 10(8):795–812, August 1999.